AN INTERACTIVE GRAPH THEORY SYSTEM*

by

Michael S. Wolfberg

Massachusetts
COMPUTER ASSOCIATES
division of
APPLIED DATA RESEARCH, INC.

29

MASSACHUSETTS COMPUTER ASSOCIATES
A DIVISION OF APPLIED DATA RESEARCH, INC.
LAKESIDE OFFICE PARK WAKEFIELD, MASSACHUSETTS 01880 (617) 245-9540

AN INTERACTIVE GRAPH THEORY SYSTEM*

by

Michael S. Wolfberg

CA-7003-0211
March 2, 1970

This is a preprint of a paper to be presented at:

Computer Graphics 70 International Symposium
April 14-16, 1970
Brunel University
Uxbridge Middlesex England

*This paper, written for the Advanced Research Projects Agency under ARPA Order Number 1228, describes work done at the Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, Pennsylvania for Rome Air Development Center and the Information Systems Branch of the Office of Naval Research under Contract NOnr 551(40).

ABSTRACT

This paper describes an interactive graphics system for solving graph theoretic problems. The system is implemented on a remote graphics terminal with processing power connected by voice-grade telephone line to a central computer. The potential of using the terminal as a programmable subsystem has been exploited, and computing power is appropriately divided between the two machines. In order to express interactive graph theoretic algorithms, the central computer may be programmed in an algorithmic language which includes data structure and associative operations. Examples of system use and programming are presented.

TABLE OF CONTENTS

INTRODUCTION

The medium of computer graphics provides a capability for dealing with pictures in man-machine communication. Graph theory is used to model relationships which are represented by pictures and is therefore an appropriate discipline for the application of an interactive computer graphics system. Previous efforts to solve graph theoretic problems by computer have usually involved specialized programs written in a symbolic assembly language or algebraic compiler language.

In recent years, graphics equipment with processing power has been commercially available for use as a remote terminal to a large central computer. Although these terminals typically include a small general purpose computer, the potential of using one as a programmable subsystem has received little attention.

These motivations have led to the design and implementation of an interactive graphics system for solving graph theoretic problems. The system operates on an IBM 7040 with a DEC-338 graphics terminal connected by voice-grade telephone line. To provide effective response times, computing power is appropriately divided between the two machines.

The remote computer graphics terminal is controlled by a special-purpose executive program. This executive includes an interpreter of a command language oriented towards controlling the existence and display of graphs. Several interactive functions, such as graph drawing and editing, are available to a user through light button and pushbutton selection. These functions, which are local to the terminal, are programmed in a mixture of the terminal computer's machine language and the interpreter command language.

For more significant computations the central computer is used, but response time for interactive operation is then diminished. In order to overcome the low bandwidth of the telephone link, the central computer may call upon a program at the terminal as a subroutine.

Based on the mathematical terminology used to define graphs, a high level language was developed for the specification of interactive algorithms. A growing library of these algorithms includes routines to aid in the construction and recognition of various types of graphs. Other routines in the library are used for computing certain properties of graphs. Graphs may be transformed by some routines with respect to both connectivity and layout. Any number of graphs may be saved and later restored.

A programmer using the terminal as an alphanumeric console may call upon the programming features of the system to develop new interactive algorithms and add them to the library. Programs may also be created for the graphics terminal, using the central computer for assembly.

ENVIRONMENT

The Interactive Graph Theory System is an experimental computer software system which operates as a user program in the environment of the Moore School Problem Solving Facility (or MSPSF) at the Moore School of Electrical Engineering of the University of Pennsylvania. The MULTILIST Project at the Moore School designed and developed the MSPSF as an attempt to combine storage and retrieval capabilities of a computer with its computational power to solve problems [3, 7, 8, 9, 10].* The hardware used for this work consists of a multi-console system attached to an IBM 7040 central processor. Since the 7040 does not directly support a variety of terminals, a small DEC PDP-8 computer is interfaced to the 7040 and is used to service Teletypes over 110-baud dial-up telephone lines and an elaborate graphics terminal via a 2400-baud private line. The graphics terminal is a DEC-338 Programmed Buffered Display, which includes a PDP-8 computer as one of two processors; the other is a more specialized processor oriented towards the control of an attached digital CRT display. Both processors share a common 8192-word 12-bit core memory with 1.5μs cycle time. Although the DEC-338 can be used as a stand-alone system it is configured as a remote computer graphics terminal, with a fixed-head minidisk of 32K 12-bit words and one DECtape for more permanent storage. The terminal includes an ASR-33 Teletype for keyboard and paper tape input and printed (and punched tape) output, a box of pushbuttons, and a light pen for graphical input.

A USER'S VIEW

A user of the Interactive Graph Theory System sits in front of the ten-inch-square display screen of the graphics terminal with the light pen in one hand and the pushbutton box conveniently placed for accessibility by his other hand. The system at the user's fingertips provides a simulated piece of "paper" on which he may draw an abstract graph. The user directs the system by using the light pen to point at light buttons and by depressing appropriate

*Numbers within brackets indicate references.

pushbuttons. Indicative messages displayed by the system instruct the user of what he may cause to happen. The novice user slowly reads these messages and carefully points at his choice of light buttons, giving the appearance that the system is controlling him. The experienced user, however, so quickly points at light buttons and depresses pushbuttons that he appears to control the system.

The user is given the tools to construct a graph of arbitrary connectivity consisting of any number of vertices and directed or undirected arcs. Any vertex or arc may be labelled, and the relative position of each label may be individually controlled. Six distinguishable shapes are available for vertices, while arcs are generally straight lines directly connecting a pair of vertices. An elliptically-shaped arc is employed when it is a loop connecting a vertex to itself. The user may view the graph on the "full" paper or he may select any one of three smaller window sizes (half- fourth-, or eighth-paper) and "move" the window to a chosen section of the paper for more detailed work. The user is given the facilities for altering or deleting any part of a graph on the paper.

Once a user has drawn a graph he may save it for future use along with any number of associated key words in the large file or data base which is available for all users of the MSPSF. Later restoration of any number of graphs may be specified by a retrieval description as a logical combination of key words.

The Interactive Graph Theory System includes a growing library of interactive graph theoretic algorithms which the user may call upon. He may, for example, choose to apply a particular algorithm to a graph which he has drawn. Figure 1 is a picture of the display screen resulting from a shortest path algorithm applied to a graph which the user drew. The integers along the directed arcs are labels the user included to indicate the cost of traversing each arc. The path computed by the system, indicated by darker arcs, is a minimum cost path from the upper vertex as the given starting point to the lower one as the given ending vertex. With this interactive algorithm, the user may request the shortest path between any pair of vertices; he may also alter any aspect of the graph and again seek a shortest path.

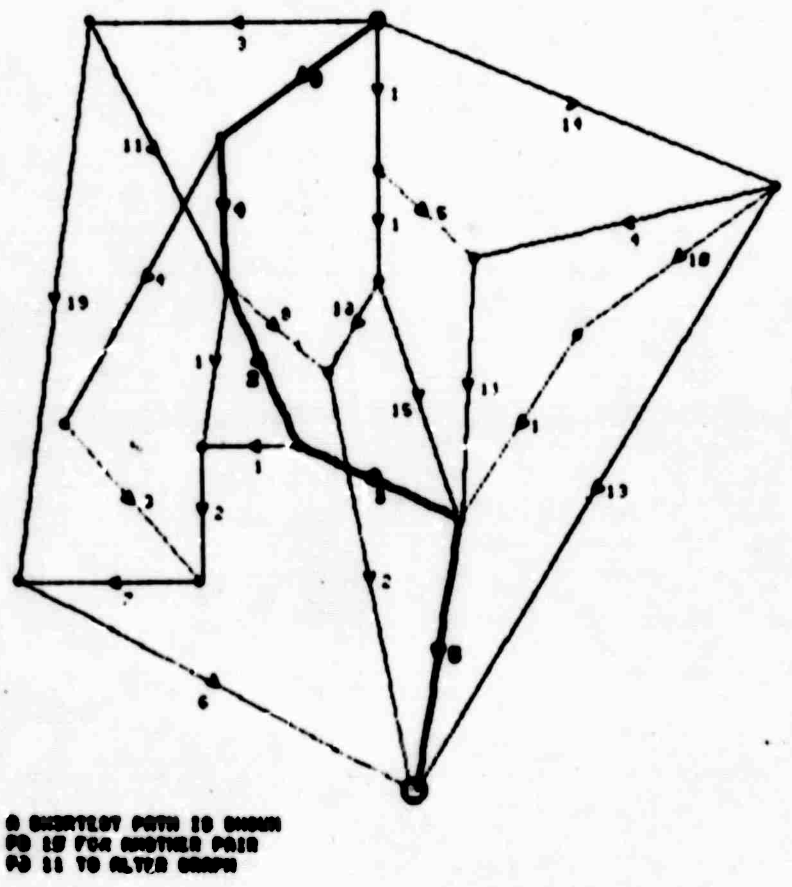


Figure 1. A Shortest Path Computed

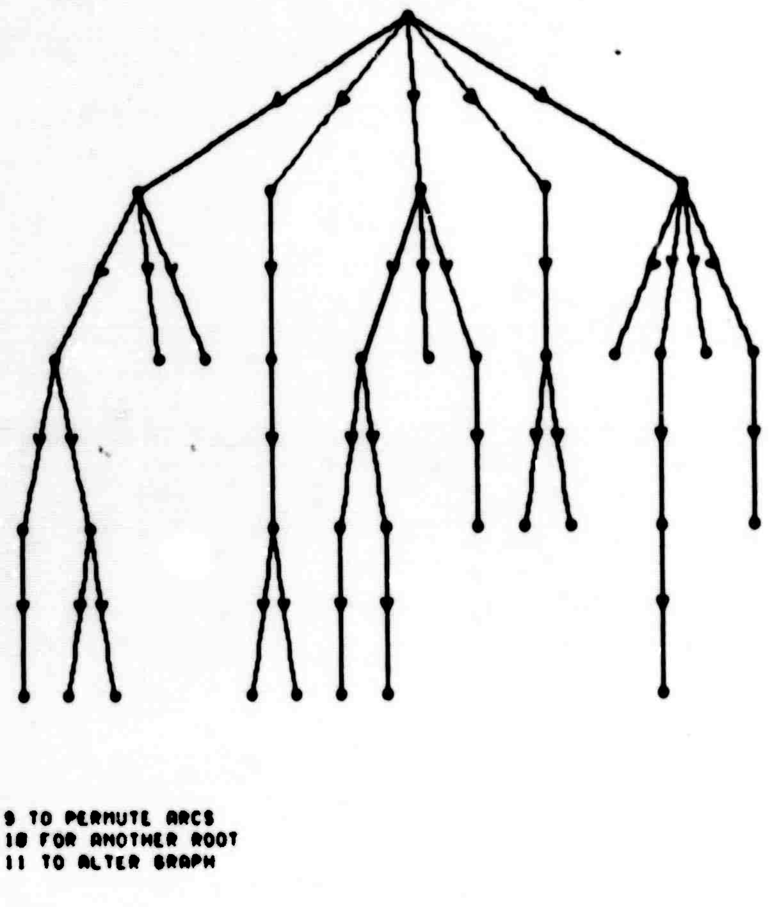
Figure 2 shows another example of the application of an interactive algorithm to a user-drawn graph. In this case the user has drawn a non-cyclic graph and then applied a layout algorithm to move the vertices so the graph appears in the form of a tree. Next, the user caused the algorithm to further refine the layout of the tree by permuting the order of the five arcs emanating from the root. In particular, the leftmost arc moved to the fourth position and the tree then appeared as in Figure 3.

Another algorithm which has been included in the system determines all maximally complete subgraphs of a given non-directed graph.* This algorithm alters the shapes of vertices to display to the user one such subgraph at a time. Figure 4 shows one maximally complete subgraph which has been computed.

These three algorithms demonstrate the types of graph theoretic problems which can be solved. The novice user can take advantage of the Interactive Graph Theory System to the extent of applying existing interactive algorithms, but the primary significance of the system is the way in which users can compose interactive graph theoretic algorithms as programs in a compiler-level language. To enable the user to develop programs, the graphics terminal can be made to operate as a text console. In this mode the DEC-338 mimics the characteristics of a Bunker-Ramo Teleregister alphanumeric console, which used to be a terminal of the Moore School Problem Solving Facility. The display screen is arranged as twelve lines of 64 characters each, and the Teletype keyboard is used to control the contents of the screen, with some of the special characters reserved for local character editing functions. One of the keyboard characters has the meaning of the TRANSMIT key of the alphanumeric console, which is to cause transmission of the contents of the display screen from the terminal to the computer. The DEC-338 is used as a text console in this system for preparing and editing programs, file manipulation, and data retrieval.

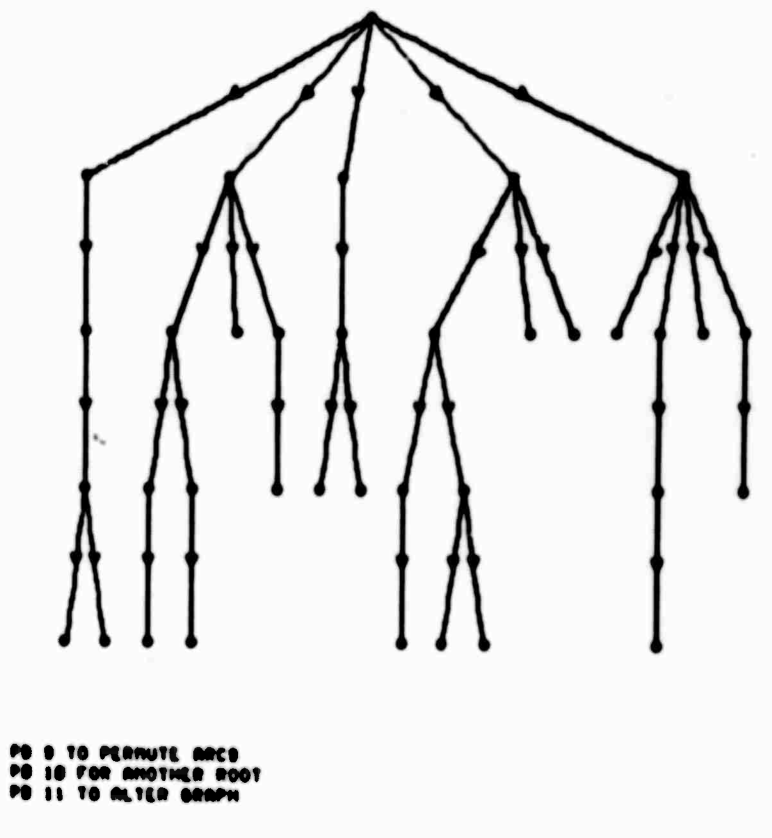
---

*A complete subgraph of a given undirected graph G is a subgraph of G such that each pair of vertices of the subgraph is connected by an arc. A maximally complete subgraph of a given undirected graph G is a complete subgraph of G which is not a subgraph of any other complete subgraph of G.



```
PB 9 TO PERMUTE ARCS
PB 10 FOR ANOTHER ROOT
PB 11 TO ALTER GRAPH
```

Figure 2. A Tree After Layout



```
PB 9 TO PERMUTE ARCS
PB 10 FOR ANOTHER ROOT
PB 11 TO ALTER GRAPH
```

Figure 3. The Same Tree with Arcs Permuted

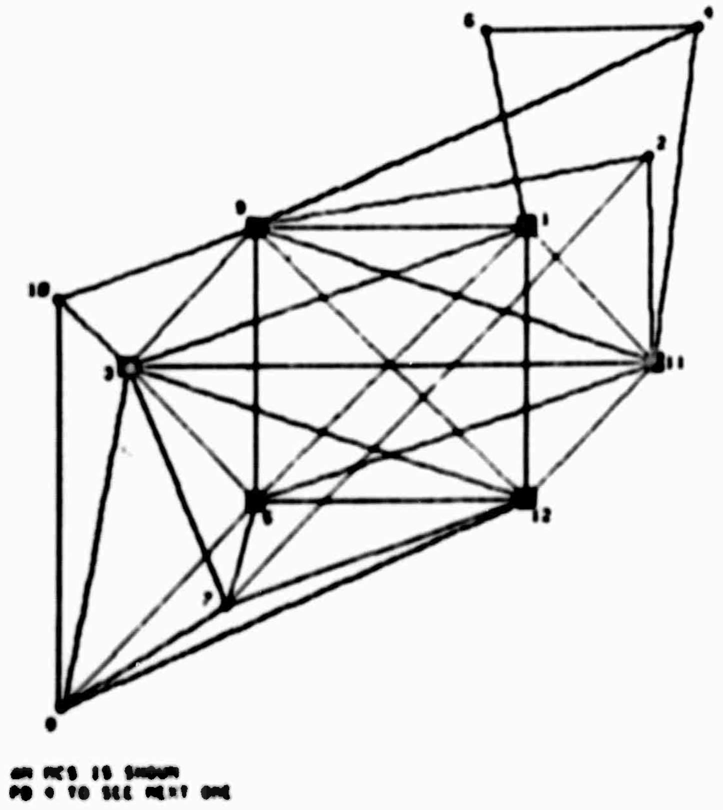


Figure 4. A Maximally Complete Subgraph Computed

THE ALLA DATA STRUCTURE AND LANGUAGE

The approach taken to represent graphs in this system was dictated by the classic definition: a graph is an ordered pair of sets (X, Γ), where X is a set of elements (vertices) and Γ is a set or ordered pairs of elements of X (arcs) [1]. The FORTRAN IV language was chosen as a base, and a new data type was introduced which would allow for the representation of ordered pairs, sets, and other appropriate constructs not representable in FORTRAN IV. Also incorporated is the facility to associate an arbitrary amount of data with any element of the data structure. The approach used was based on that used by George Dodd in the design of the Associative Programming Language as an extension of PL/I [2]. This extension of FORTRAN is called ALLA. One deficiency of Dodd's APL which has been eliminated in ALLA is the necessity for specifying in advance the allowable associations an entity may have.

The additional data type which has been introduced is named entity. There is no literal naming of entities in ALLA except for the undefined entity UNDEF. Instead, entities are referenced through entity variables or by a relation or association with an entity. There are three types of entities: atom, pair, and set. Each declared entity variable may, at any one time, name a particular atom, pair, or set, or it may have a value of UNDEF.

Each atom, pair, or set may have any amount of associated data. An associated datum is called a property and is referenced by a property name. The value of the property of an entity may be an integer, real, or logical constant, or it may be an entity.

An entity of the type atom is one which has no structure other than its associated properties. A pair is a type of entity which, in addition to any properties, has a left-element and a right-element, each of which may be an entity or may be undefined.

A set, besides having associated properties, is a structure with any number of elements, each of which must be an entity. A set may have no elements, in which case it is called empty. Although the word "set" is used, the implementation imposes an ordering to the elements, and so one may make use of the "list" nature of this structure. Also, membership in a set is not limited to a particular element appearing once. There are no restrictions on the structuring of data in this system. For example, a particular set may even be a member of itself three times. More important, however, is the unlimited hierarchy of the relationships which can be modelled in this structure.

With this data structure a graph will be defined as an ordered pair, where the left-element of the pair is the set of vertices, and the right-element is the set of arcs. Each arc is an ordered pair, where the left element is the "from-vertex" of the arc, and the right element is the "to-vertex". Each vertex will ordinarily be an atom, but the ALLA data structure permits any type of entity as an element of a pair or set. Thus one could even represent a graph of graphs, or other interesting structures. More commonly, one finds the following structures appearing in graph theory manipulations:

1. A set of arcs (where order is important) for a path.
2. A set of arcs as an entity property of a vertex for its outgoing arcs.
3. A set of vertices as an entity property of a vertex for its neighbors.
4. An integer property of a vertex for its depth in a tree.

EXAMPLE OF A GRAPH THEORETIC ALGORITHM

Instead of presenting a full description of ALLA, a practical example will be explained in detail. The SHPTHW function chosen for this illustration is the algorithm for finding a <u>sh</u>ortest <u>p</u>a<u>th</u> in a <u>w</u>eighted directed graph. It was this function, used along with an interactive ALLA routine to interface with the user, which was responsible for producing Figure 1. The reader should refer to the source listing of SHPTHW in Figure 5 in the following discussion. The lines of the function have been numbered for reference.

Figure 5 constitutes the definition of the function SHPTHW whose value or result is a shortest path given a graph G and the starting vertex A and

```
LINE

  1          ENTITY FUNCTION SHPTHW(A,B,G)
  2          ENTITY A,B,G,DSET,TDSET,V,OAV,RVOAV,AA
  3          ENTITY INARC,OUTARC
  4          INTEGER WEIGHT,DIST
  5          PROPERTY DIST
  6          THROUGH 10 FORALL V IN LELM(G)
  7     10   DIST(V) = 10000000
  8          DIST(A) = 0
  9          INSERT A INTO CRSET(DSET)
 10     20   TDSET = DSET
 11          CREATE SET DSET
 12          THROUGH 40 FORALL V IN TDSET
 13          THROUGH 30 FORALL OAV IN OUTARC(V)
 14          RVOAV = RELM(OAV)
 15          IF (DIST(RVOAV) .LE. (DIST(V) + WEIGHT(OAV))) GOTO 30
 16          DIST(RVOAV) = DIST(V) + WEIGHT(OAV)
 17          INSERT RVOAV INTO DSET
 18     30   CONTINUE
 19     40   CONTINUE
 20          DELETE TDSET
 21          IF (.NOT.EMPTY(DSET)) GOTO 20
 22          DELETE DSET
 23          CREATE SET SHPTHW
 24          IF (DIST(B) .EQ. 10000000) GOTO 300
 25          V = B
 26    110   IF (V .EQ. A) GOTO 300
 27          THROUGH 120 FORALL AA IN INARC(V)
 28    120   IF ((DIST(LELM(AA))+WEIGHT(AA)).EQ.DIST(V)) GOTO 130
 29          CALL ERROR(6HSHPTHW)
 30    130   INSERT AA INTO SHPTHW
 31          V = LELM(AA)
 32          GOTO 110
 33    300   RETURN
 34          END
```

Figure 5. Shortest Path Function

ending vertex B. The result of the function is a set of arcs of the given graph; if no path is possible, the set is empty. Line 1 is the function declaration. It is an entity function since its result is an entity, i.e. a set of arcs. Lines 2 through 4 declare the data types of the three arguments of the function and all of the variables used within the function body, both locals and externals. Line 5 declares DIST as a property name.

The SHPTHW function assumes that the given graph is of the form described in the previous section, and associated with each vertex is its set of incoming arcs (INARC) and its set of outgoing arcs (OUTARC). Also associated with each arc of the given graph is its integer weight (WEIGHT). Throughout the body of the function the uses of the names INARC, OUTARC, and WEIGHT are purposely ambiguous. That is, the ALLA syntax is ambiguous, so that each of these three names may be either properties or functions; only the environment determines which is the case. Thus the SHPTHW function assumes either the given graph already possesses values of the three properties, or there are equivalent functions which compute them.

The program, which employs Moore's algorithm [6], begins on lines 6 and 7 by assigning an associated distance of infinity (actually ten million here) to each vertex in the given graph. The THROUGH statement on line 6 specifies that the entity variable V should on each iteration of the range of the THROUGH loop assume the value of the next element of the set LELM(G). The term LELM(G) represents the left-element of the pair G, which is the set of vertices of the given graph. On line 8 the starting vertex is given an associated distance of 0. The property DIST is being used to represent the minimum known path cost from the starting vertex. Initially, the distance assigned to the starting vertex is trivially known to be 0. All other distances are assumed to be infinite, since without considering the graph's connectivity all other vertices are potentially unreachable.

The algorithm consists of an iterative search on lines 9 through 21 followed by a reverse trace. The search begins at the starting vertex and a distance is assigned to each vertex which can be reached from the starting vertex by traversing one outgoing arc. Next, a distance is assigned to each "neighbor" vertex which is connected by an outgoing arc to one of the vertices just assigned. The distance of each neighbor vertex is equal to the distance of its previous vertex plus the weight of their connecting arc. If a distance has already been assigned to a vertex, it is replaced with a new distance only when the new distance is numerically smaller. This process is repeated until a pass is made which does not improve any distance in the graph. At this time, the distance associated with each vertex in the graph is the minimum cost to reach that vertex from the given starting vertex.

The statement on line 9 should be read as "insert the element named by entity variable A into a created set henceforth named by entity variable DSET". The statement on line 10 has entity variables on both sides of the equal sign. Whereas FORTRAN semantics dictate a copy would be made if these were integer, real, or logical variables, the entity variable TDSET is made to reference (or name, or point to) the entity referenced by DSET. In general, this interpretation applies to entity expressions on the right side of the equal sign, as occurs on line 14.

During each iteration of the search, the entity variable TDSET references the set of previously assigned vertices. DSET names the set used to keep track of the new vertices being assigned during an iteration. Line 21 contains the test for whether another iteration is needed. EMPTY is a predicate function which is .TRUE. when its argument is a set with no members.

Note that the ALLA programmer is required to perform his own management of storage: on each iteration the DELETE statement on line 20 causes the freeing of the space used to model the set referenced by TDSET.

After the last iteration there is a check at line 24 to ascertain if the given ending vertex has been assigned a distance; if its distance has remained infinite it cannot be reached, and there is no trace. Otherwise, the trace starts with the ending vertex at line 25. Each incoming arc is considered along with the vertex from which the arc emanates. If the distance of that vertex plus the weight of that arc equals the distance assigned to the ending vertex, that arc is part of a shortest path. This process continues until the statement on line 26 determines the starting vertex has been reached, at which time the answer has been computed as the set of arcs SHPTHW.

IMPLEMENTATION OF ALLA

The ALLA data structure and language which has been introduced above is implemented on two modular levels. First, the compilation of the language is effected by preprocessing all of the non-FORTRAN statements into FORTRAN subroutine and function calls. The package of subroutines which constitutes the run-time system to realize the ALLA data structure is the second level of implementation. In order to provide for machine independence, and allow for easier writing, debugging, and modification, the L^6 language was selected for both the implementation of the ALLA data structure into a particular memory structure, and the preprocessing of ALLA into FORTRAN. L^6 was originally designed and implemented at the Bell Telephone Laboratories where it was named "Bell Telephone Laboratories' Low-Level Linked List Language" or L^6 (pronounced "L-six") [5]. Based on the original implementation on the IBM 7094, the author implemented UP.L6 for the IBM 7040. In the process of translation, improvements and new features were added, including the facility of linking L^6 programs with both FORTRAN and MAP assembly language subroutines [12].

The separation of data structure (the ALLA programmer's view) from the implementation of memory structure (the systems programmer's view) is most valuable, especially when the language used to implement memory structure is of the level L^6. This modularity supports the freedom to reorganize the memory structure at any time in order to adjust time-space operating characteristics. Also, a researcher may use such a framework for comparative studies of memory structures. For example, a doubly-linked list approach might be compared against the utilization of lists with only forward pointers.

ROLE OF THE GRAPHICS TERMINAL

A common concern of designers of computer graphics systems is the choice of data and memory structures not only to model relationships among the data, but also for the maintenance of the display of the data. When a single computer system is used for the implementation of a computer graphics system, the tendency is to employ one structure which can also be used to drive the display controller. However, when the graphical equipment resides at a remote site along with a small computer linked to the central computer by voice-grade telephone line, at least a graphics-oriented structure must be maintained at the terminal for effective interaction.

One possible solution to the need for two structures in two (often) different computers is the implementation of the same structure in both machines. Perhaps the terminal computer would handle a subset of what the central computer can do. A strong motivation for such an approach is the capability for having programs which can operate in either or both machines. Although this is potentially powerful, unless the two computers are appropriately related, it could be stifling to the effectiveness of both machines. Namely, the structure in the smaller one might be so general that its small capacity is too quickly exceeded, and, at the same time, the possibility for sophistication in the larger machine might be suppressed.

The advantages of modularity have dictated the division of labor employed in the Interactive Graph Theory System. A special-purpose structure in the graphics terminal contains only the parameters relevant to the display of graphs. This structure and its associated display file are managed by a special-purpose executive program named <u>DOGGIE</u>, for <u>D</u>isplay <u>o</u>f <u>G</u>raphics <u>G</u>raphical <u>I</u>nterpretive <u>E</u>xecutive, which controls the DEC-338 terminal. As its name indicates, its primary role is the interpretation of a special-purpose command language. DOGGIE commands are scanned by the interpreter as sequences of 12-bit bytes which may either be received from <u>user programs</u> running in the DEC-338 itself. 1600_8 locations of the DEC-338 are reserved for the execution of these user programs which consist of a mixture of PDP-8 machine language and DOGGIE command language. A programmer-user of this system may easily avoid the need for knowledge of the DEC-338 and restrict his programming efforts to the interactive ALLA language. However, the facilities are available for anyone to write his own user programs. User program preparation is performed using the terminal as a text console. The assembly of DEC-338 programs is carried out on the IBM 7040 by the PDPMAP Assembly System [4], which is significantly superior to using the DEC-338 itself.

All DOGGIE commands implicitly refer to a scratchpad "paper" on which a single graph may be defined. It is only one graph in the sense that there is no facility for hierarchical grouping; however, the single graph may consist of any number of disjoint components which may give the effect of displaying more than one graph at a time. The graph may consist of any number of vertices or arcs, and it may be only partially defined at any time, since it is permissible to define an arc in terms of vertices which do not yet exist. There is separate control over which parts of the defined graph are to be displayed.

The graph maintained by DOGGIE is built, modified, and deleted through interpreted DOGGIE commands. The command language includes elements which affect the gross aspects of the existing graph or the way in which the graph is being displayed. A group of commands may refer to a unique vertex or arc by internal name or to all existing vertices or arcs. The option is also available for DOGGIE to supply a created internal name when a vertex or arc is defined. A list of other services performed by DOGGIE follows:

1. manages all input/output of the DEC-338 by handling interrupts from the various display flags, the Dataphone interface, the minidisk, and the Teletype.
2. manages the display of the graph on the "paper" with four window sizes available for viewing all or any part of the paper.
3. performs light pen tracking with optional horizontal and/or vertical constraints on a pseudo-pen-point.
4. interprets light pen hits as a result of a user's pointing at displayed parts of a graph.
5. helps in the management of the pushbutton box.

6. handles overlays of user program segments by name by interfacing with the PDP-8 Disk Monitor System.

7. collects and maintains status information concerning the state of DOGGIE and its existing graph.

Note that this system design gives the programmer explicit control of what is displayed instead of automatically monitoring the ALLA structure. Also, the division of labor employed makes it feasible to substitute another computer at either end of the telephone line. For example, an equivalent DOGGIE executive and set of user programs could be implemented on an ADAGE terminal.

INTERACTIVE PROGRAMS

A user program which operates at the terminal is appropriate for problems where nearly instantaneous system response is important. An interactive algorithm may be entirely resident in the DEC-338, it may be entirely resident in the IBM 7040, or it may be divided between the two computers. There are facilities to shift the center of control from one computer to the other. For example, an interactive ALLA program running in the IBM 7040 may call upon a user program in the DEC-338 as a subroutine.

An interactive algorithm written for execution at the terminal may be appropriate only if minor computing is required and if associated data can be conveniently represented within the framework of the terminal's data structure. Although this is not the recommended method for implementing graph theoretic algorithms, the power of the small machine has been demonstrated by implementing a user program which interprets a Mealy state graph prepared in a prescribed format and carried out the operations of a finite state acceptor. The "current" state is indicated by a blinking vertex, and the user supplies an input character from the Teletype keyboard (or paper tape reader). According to the typed (or read) character, an output character (or string) is immediately typed out on the Teletype printer, and the new current state is made to blink.

The set of DOGGIE commands constitutes a machine-independent language for controlling the display of graphs. The language in the pure sense is not interactive since it is only an output language. It becomes interactive when used in conjunction with other languages which include control specification. The DOGGIE language is used in two different environments within the Interactive Graph Theory System: first, it is embedded into ALLA language for execution in the IBM 7040. Second, it is embedded into PDPMAP Assembly Language through macros for use in user programs operating in the DEC-338. The use of the DOGGIE language in either environment has the same meaning, which is to direct the DOGGIE interpreter to perform commands which define, alter, and display graphs.

The writing of interactive programs is somewhat different in the two environments, but the basic idea is common to both languages. The input aspect of the interaction is accomplished by programming in the host language the observation of communication cells. These cells reflect the status of the DEC-338 and DOGGIE back to the programmer. The information contained in these cells includes:

1. current graph display status - intensity, window size and position,
2. light pen tracking indicators,
3. indication of the amount of available free blocks,
4. light pen hit information, and
5. complete status output for a vertex or arc.

A communication cell is a rather natural concept for use in the DEC-338, for in that environment it is simply an accessible memory location. User programs operating in the DEC-338 are written with references to the symbolic names of these cells. There are some communication cells in the DEC-338 which are used as indirect addresses of subroutines included within DOGGIE such as the subroutine to send an 8-bit character over the Telephone line. A number of communication cells of this type are not duplicated in the ALLA environment.

Communication cells in the ALLA environment are referenced as any other FORTRAN integer variable or logical variable. They are automatically declared in each ALLA subprogram, so the programmer simply references these cells by symbolic name.

Many of the communication cells in the 7040 are essentially copies of the "real" cells in the DEC-338. The "real" introduced here means real-time. For example, a pair of communication cells indicates the position of the pseudo-pen-point linked to the light pen tracking cursor. In the DEC-338, user programs observing these cells may do so in real-time. However, the communication cells of the DEC-338 are copied over to the 7040 only when requested by certain statements in interactive ALLA programs. Since this copying operation takes about one or two seconds to complete, and since tracking may alter the position of the pseudo-pen-point every few milliseconds, the communication cells in the 7040 cannot reflect this data in real-time.

EXAMPLE OF AN INTERACTIVE PROGRAM

An example of a rather small, yet complete interactive ALLA program is presented in Figure 6. The following discussion references the SAMPLE subroutine in Figure 6, which assumes a graph is already being displayed at the terminal. On line 2, the subroutine call causes the clearing of the flag associated with pushbutton number 11 on the pushbutton box at the terminal. Line 3 contains a DOG statement which indicates the remainder of the line is a symbolic DOGGIE command. This symbolic form of DOGGIE command causes two 12-bit bytes to be sent to the DEC-338: octal 3600 and 0000, which the DOGGIE interpreter will interpret as a command to make light pen sensitive all vertices currently existing in the graph. Next, the statements on lines 4 through 7 cause a message for the user to be placed on two message lines at the lower left corner of the terminal's display screen. As the text of the message indicates, this sample program allows the user to force the shape of any vertex "seen" by the light pen to be made square (vertex shape 6) until pushbutton 11 is depressed. The statement on line 8 causes light pen hits to be allowed. The WAITCHANGE statement of line 9 shifts control of the system to the terminal until some change occurs in the status of pushbuttons,

```
LINE

 1                SUBROUTINE SAMPLE
 2                CALL CLRPB(11)
 3                DOG START LTPEN WHOLE VERTEX, ALL
 4                CALL MESSAG(2)
 5                DOGSTRING 'POINT TO VERTICES TO BE'
 6                CALL MESSAG(1)
 7                DOGSTRING 'MADE SQUARE, PB 11 TO STOP'
 8        10      DOG ALLHIT
 9        20      WAITCHANGE
10                IF (PB(11)) GOTO 30
11                IF (LPHIT1 .EQ. 0) GOTO 20
12                DOG START EXIST SHAPE VERTEX 6,(LPHIT2)
13                GOTO 10
14        30      TERMINATE
15                STOP
16                END
```

Figure 6. Sample Interactive Program

light pen, or Teletype input. When a change occurs, communication cells in the ALLA environment are updated and control shifts to line 10 where the current status of pushbutton 11 is checked. If it has not been depressed by the user, control flows to line 11 where the communication cell LPHIT1 is checked for a value of 0. A non-zero value indicates a light pen hit occurred, and the communication cell LPHIT2 contains the internal name of the vertex which caused the hit. In this case, the statement on line 12 causes the shape of the hit vertex to be square.

SUMMARY

This paper has described how a remote computer graphics terminal with processing power is used in a multi-console operating system as an alphanumeric console and an interactive graphics device. An Interactive Graph Theory System was built in this environment to exhibit the effective use of such a terminal and to demonstrate a design for a programming system for solving graph theoretic problems.

In order to express interactive graph theoretic algorithms, the central computer's FORTRAN IV language has been enriched with data structure and associative operations and a class of statements to control the existence and display of graphs at the terminal. The implementation of this language employs a separate module to specify the underlying memory structure using L^6.

The terminal computer is managed by an executive program which incorporates an interpreter of a special-purpose command language oriented towards controlling the existence and display of graphs. It may be programmed to carry out local functions as well as those which are performed as subroutines of an interactive program running in the central computer.

Examples of system use and programming in the interactive ALLA language have been presented.

All of the work described in this paper is fully reported in the author's Ph.D disseration [11], which includes complete programming manuals of interactive ALLA and DOGGIE command language. Copies are available from the author.

REFERENCES

1 C BERGE
The theory of graphs and its applications
John Wiley and Sons NY 1964

2 G G DODD
APL - a language for associative data handling in PL/I
Proc FJCC 1966 677-684

3 D K HSIAO
A file system for a problem solving facility
Dissertation in EE Univ of Pa 1968

4 T H JOHNSON M S WOLFBERG
The PDPMAP assembly system
Moore School of EE Report 68-11 Univ of Pa 1967

5 K C KNOWLTON
A programmer's description of L^6
CACM Vol 9 No 8 1966 616-625

6 E F MOORE
Shortest path through a maze
Annals of the C [illegible]tation Laboratory of Harvard Univ
Vol 30 Harvard Univ Pres 1959

7 R P MORTON
On-line computing with a hierarchy of processors
Dissertation in EE Univ of Pa 1968

8 R P MORTON M S WOLFBERG
The input/output and control system of the moore school problem solving facility
Moore School of EE Report 67-30 Univ of Pa 1967

9 N S PRYWES
Man-computer problem solving with multilist
Proc IEEE 1966 1788-1801

10 R L WEXELBLAT
The development and mechanization of a problem solving facility
Dissertation in EE Univ of Pa 1965

11 M S WOLFBERG
An interactive graph theory system
Dissertation in EE Univ of Pa 1969
also Moore School of EE Report 69-25 Univ of Pa 1969

12 M S WOLFBERG P A T WOLFGANG
UP.L6 - an L^6 system for the IBM 7040
Moore School of EE Univ of Pa 1966 Internal report